TRABALHO DE CONCLUSÃO DE CURSO

Aplicação de Redes Neurais para codificação rápida de vídeo usando o H.265/HEVC

Tomás Rosário Rosemberg

Brasília, Novembro de 2019

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

Aplicação de Redes Neurais para codificação rápida de vídeo usando o H.265/HEVC

Tomás Rosário Rosemberg

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro da Computação

Banca Examinadora

Prof. Eduardo Peixoto Fernandes da Silva, Ph.D, FT/UnB
Orientador _____

Prof. Alexandre Ricardo Soares Romariz, Ph.D, FT/UnB
Examinador interno _____

Prof. Tiago Alves da Fonseca, Dr., FGA/UnB
Examinador interno _____

FICHA CATALOGRÁFICA

ROSEMBERG, TOMÁS ROSÁRIO
Aplicação de Redes Neurais para codificação rápida de vídeo usando o H.265/HEVC [Distrito Federal] 2019.
xvi, 28 p., 210 x 297 mm (ENE/FT/UnB, Engenheiro, Engenharia Elétrica, 2019).
Trabalho de Conclusão de Curso - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. HEVC	2. Deep Learning
3. H.265	4. Compressão
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

ROSEMBERG, T.R. (2019). *Aplicação de Redes Neurais para codificação rápida de vídeo usando o H.265/HEVC*. Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 28 p.

CESSÃO DE DIREITOS

Tomás Rosário Rosemberg
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Agradecimentos

Meus agradecimentos aos amigos, tanto os feitos durante esse percurso da faculdade quanto os de fora dela. Companheiros de trabalhos e inúmeras noites não dormidas para que fosse possível concluir essa etapa de nossas vidas. Agradeço minha família por todo o apoio incondicional nessa longa caminhada de grandes aprendizados e preparação para novas jornadas.

Tomás Rosário Rosemberg

RESUMO

A compressão de vídeo é uma área de estudo em constante desenvolvimento, com a grande demanda de consumo de vídeos, os quais evoluem em termos de tamanho e qualidade, o que demanda constante melhoria. Desta forma ela recebe diversas propostas de incorporação de estudo de outras áreas, visando agregar conhecimento e resultado. Embora alguns codificadores padrão, como o H.265/HEVC, ofereçam um excelente desempenho de taxa-distorção, eles o fazem com um alto custo computacional.

Ao mesmo tempo, a inteligência artificial tem atraído cada vez mais atenção. Neste trabalho propomos a incorporação da Inteligência Artificial como meio de acelerar a compressão baseada no padrão H.265/HEVC, utilizando redes neurais para tentar predizer uma tomada de decisão que ocorre neste formato. Compara-se os resultados obtidos usando redes neurais desenvolvidas com o framework *Tensorflow* [1] usando a linguagem de programação *Python*, com o modo padrão do codificador HM 16.20 respeitando as regras do CTC. Como questão adicional são propostas possibilidades de melhorias para futuro desenvolvimento.

ABSTRACT

Video compression is a constantly evolving field of study, with the high demand for video consumption, which evolves in terms of size and quality, it needs to be constantly improving. So it receives several proposals for incorporation from other areas, expecting to aggregate knowledge and results. Although some compressors, like the H.265/HEVC, achieves a great result for the distortion rate, it does have high computational cost.

Simultaneously, Artificial Intelligence is attracting more and more attention. In this work, we propose the incorporation of Artificial Intelligence as a way of accelerate compression based on the H.265 / HEVC standard, using neural networks to try to predict decision that is taken this format. The results obtained using neural networks created with the *Tensorflow* [1] structure using a *Python* programming language, are compared with the default mode of the HM 16.20 encoder, respecting the CTC rules. As an additional issue is proposed the possibility of improving the work for future development.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

CTC	*Common Test Conditions* (Condição de Teste Comum)
CTU	*Coding Tree Unit*
CU	*Coding Unit* (Unidade de Compressão)
GB	*Gigabytes*
HEVC	*High Efficiency Video Coding* (Compressor de Vídeo de Alta Eficiência)
HVS	*Humna Visual System* (Sistema Humano de Visão)
MSE	*Mean Square Error* (Erro Quadrático Médio)
PSNR	*Peak Signal-to-Noise Ratio* (Relação Sinal-Ruído de Pico)
QP	*Quantization Parameter* (Parâmetro de Quantização)
RAM	*Random Access Memory* (Memória de Acesso Randômico)
RELU	*Rectified Linear Unit* (Unidade de Retificação Linear)
RGB	*Red Green Blue* (Vermelho Verde Azul)
SVM	*support vector machine* (Máquina de Vetores de Suporte)

1 INTRODUÇÃO

Os vídeos estão cada vez mais presentes na vida do ser humano. Relatórios indicam que eles devem compor 82% do tráfego global da Internet até 2022 [2]. Porém, a utilização de vídeos na sua forma pura se torna inviável pois um vídeo de resolução 1920px $\times$ 1080 px pode atingir 1,6GB para apenas 10s, desta forma seriam necessários 216GB para armazenar uma hora de vídeo nessas condições, tornando inviável a sua utilização em formato puro no cotidiano das pessoas, visto que encontramos discos de DVD para reprodução doméstica de vídeos com capacidade de 4,7GB no mercado, e mesmos os discos de *blu ray* tem 50 GB de capacidade. Desta forma podemos perceber que não é prático o armazenamento do vídeo em seu formato puro nestes discos.

Dado esse fato, existem áreas de estudo sobre como comprimir essa informação de forma viável para o uso, com o mais novo padrão utilizado sendo o H.265/HEVC [3]. Um problema é que a complexidade computacional deste padrão é alta, requerendo uma boa capacidade computacional para sua utilização e tempo. Embora o codificador de referência HM [3] não seja otimizado para aplicações em tempo real, ao utilizar o codificador HM 16.20 [4] no seu modo padrão, durante esse trabalho, necessitamos de 23mil segundos, ou seja aproximadamente 6 horas para poder comprimir um vídeo de 10 segundos.

Ao mesmo tempo, técnicas de aprendizado de máquina estão presentes em inúmeras tecnologias comerciais modernas, com consideráveis avanços recentes no desempenho destes algoritmos utilizando redes neurais. Esta tecnologia vem impactando diariamente a vida do consumidor com suas mais diversas aplicações, sendo utilizada em aplicações de reconhecimento de imagens, transcrição de texto, sugestão de produtos em sites de *e-commerce* e filmes, vídeos e músicas em plataformas de *streaming*, direcionamento de conteúdo em redes sociais e resultados de busca relevantes em plataformas de pesquisa [5]. Com avanços significativos nos últimos anos, o uso de redes neurais apresentou melhora consideravel em múltiplas áreas de estudo, definindo o novo estado da arte em reconhecimento de voz, reconhecimento visual de objetos, descobrimento de drogas e genômica [5].

Neste trabalho será abordada a aceleração da compressão de vídeos aplicando redes neurais ao H.265/HEVC. A ideia é treinar redes neurais que se baseiem nos pixels das imagens e consigam predizer como será feita a partição dos blocos, desta forma diminuindo a quantidade de combinações de formato de blocos possíveis, reduzindo o tempo necessário para compressão sem diminuir significativamente a qualidade da compressão.

Para exploração dos resultados, utilizamos a *common test condition* (CTC) para garantir a replicabilidade e confiabilidade do método proposto, e comparamos os resultados obtidos com o padrão do compressor HM 16.20 e um algoritmo que prevê a partição dos blocos de modo aleatório.

Tem-se como objetivo deste trabalho desenvolver uma rede que apresente um desempenho bom no aspecto de prever a partição dos blocos e possibilite um menor tempo necessário para comprimir os vídeos usando o H.265/HEVC. Também espera-se que os resultados aqui demonstrados possam servir de base para estudos futuros que possam partir do princípio fundamental deste trabalho, ou de ideias comentadas no Capítulo 6 , que traz ideias a respeito de futuros desenvolvimentos.

2 APRENDIZADO DE MÁQUINA

Aprendizado de máquina é a ciência de programação de computadores para que eles possam detectar padrões a partir de dados[6]. Por exemplo, um filtro de spam de e-mail é um programa de aprendizado de máquina que aprende a diferenciar e-mails normais de spams baseando-se em marcações de usuários.

Com a utilização de técnicas de aprendizado de máquina a abordagem para a resolução do problema costuma ser diferente. Enquanto a abordagem mais tradicional costuma ser estudar o problema, escrever as regras para a resolução do problema e avaliar o desempenho, a abordagem de aprendizado de máquina é diferente, visto que estas regras podem ser bastantes complexas para o ser humano conseguir escrevê-las. Nesta abordagem há o estudo do problema, porém no momento de escrever as regras, substituímos esse passo por, com a utilização de algoritmos e dados, utilizar o computador para que ele encontre essas regras, e enfim avaliar o desempenho. Existem diversos tipos de sistemas do aprendizado de máquina, sendo alguns deles o aprendizado supervisionado e o aprendizado não supervisionado.

O aprendizado supervisionado, como ilustrado na Figura 2.1, baseia-se na informação prévia de rótulo dos dados. Nele utilizamos os dados, também chamados de conjunto de treinamento, como entrada para o programa. Sabendo previamente a saída desejada, treinamos o computador para que ele possa identificar um padrão nesse conjunto de dados, para que depois, ao avaliar dados sem esses rótulos, ele seja capaz de fornecer a saída desejada para nós. Um típico uso de aprendizado supervisionado são algoritmos de classificação, onde o programa é usado para classificar a entrada dentro de classes. Outro famoso uso de aprendizados supervisionado é o algoritmo de regressão linear, onde o programa procura identificar qual a melhor reta que descreve aqueles dados de treinamento, para assim, quando inserirmos novos dados sejamos capaz de obter um valor para ele próximo a realidade.

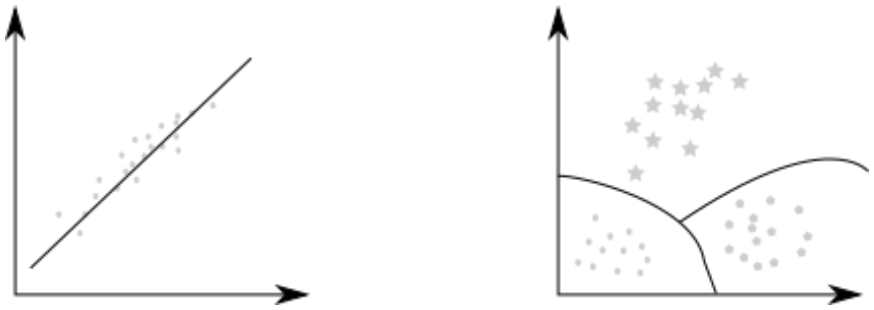

Figura 2.1: A esquerda um exemplo de aprendizado supervisionado (regressão linear) e a direita um não supervisionado (Clustering).

O aprendizado não supervisionado, também ilustrado na Figura 2.1, não possui os rótulos dos dados previamente. Para este tipo de programa, os algoritmos tentarão identificar padrões sozinhos, podendo ser usado para identificar anomalias no seu conjunto de dados ou separar em

classes o conjunto de dados.

Problemas de classificação são um típico problema onde se é empregado aprendizado de máquina, eles são problemas onde desejamos desenvolver ferramentas que, tendo estudado um grande conjunto de dados já catalogados, deve conseguir catalogar dados não vistos anteriormente [7]. Eles são bem comuns e possuem diversos algoritmos de possível utilização para resolvê-los, como por exemplo classificadores de Naïve Bayes, redes neurais, árvore de decisão e SVMs (máquina de vetores de suportes) onde cada um deles possuem seus prós e contras [8].

2.1 REDE NEURAL ARTIFICIAL

A base para criação das redes neurais artificiais vem do estudo acerca do funcionamento do cérebro humano. Ao analisarmos a unidade base do cérebro humano, identificamos a presença do neurônio, formado pelos dendritos, o núcleo, o axônio e suas terminações. A informação chega ao neurônio pelos seus dendritos, passa pelo axônio até chegar às suas terminações, onde as sinapses entre o neurônio e o próximo ocorrem, perpassando, ou não, a informação. Na intercomunicação dos neurônios existe diferença de influência de um neurônio no próximo, o que foi modelado de forma simplificada pela primeira vez por McCulloch e Pitts em 1943 [9] e pode ser descrito como

$$y = f(z), \textit{tal que } z = \sum_{i}^{N} w_i x_i + b, \tag{2.1}$$

onde a saída de um neurônio é dada a partir de uma função de ativação baseada em suas entradas, x_i são as entradas do neurônio, w_i são os pesos, demonstrando quanto cada entrada influencia no resultado e b é o viés. Esta saída do neurônio é depois passada por uma função de ativação.

2.1.1 Unidade Básica

O neurônio básico para o estudo de redes neurais pode ser representado pela Equação 2.1, sendo a soma ponderada dos sinais de cada conexão de entrada somada a um viés. Ele é designado como nó. Este nó receberá valores de entradas e fornecerá uma saída a partir deles. A saída será descrita a partir do somatório destes valores de entradas multiplicados por quanto estes valores devem influenciar o resultado final, as variáveis que definem o quanto cada entrada influencia o resultado final é chamado de peso. Soma-se a este valor um viés. Após isso tudo, este resultado é passado por uma função de ativação para fornecer a saída do nó. A saída deste nó poderá ser o resultado final de uma rede, ou a entrada para próximos nós, dependendo da arquitetura da rede.

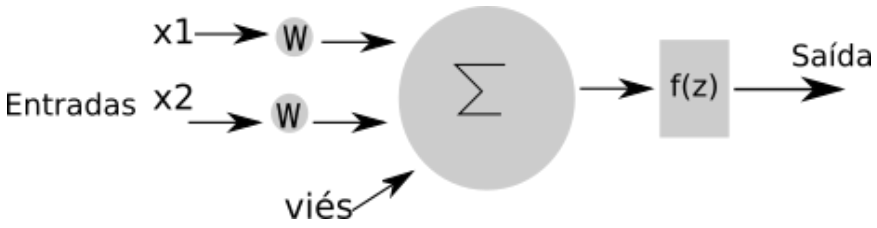

Figura 2.2: Perceptron

2.1.2 Múltiplas Camadas

A utilização de um único neurônio possui limitações para conseguir representar funções mais complexas, desta forma ocorre a junção de vários nós em camadas para a criação de um rede neural artificial de múltiplas camadas. Estas redes neurais acabam possuindo três parte fundamentais: a entrada, onde se recebe a entrada de informações a serem processadas, a camada oculta, com a existência de nós distribuídos em camadas, onde as saídas dos nós de uma camada serão a entrada dos nós da próxima camada, e por último, a camada de saída, onde ocorre o resultado da análise da rede neural.

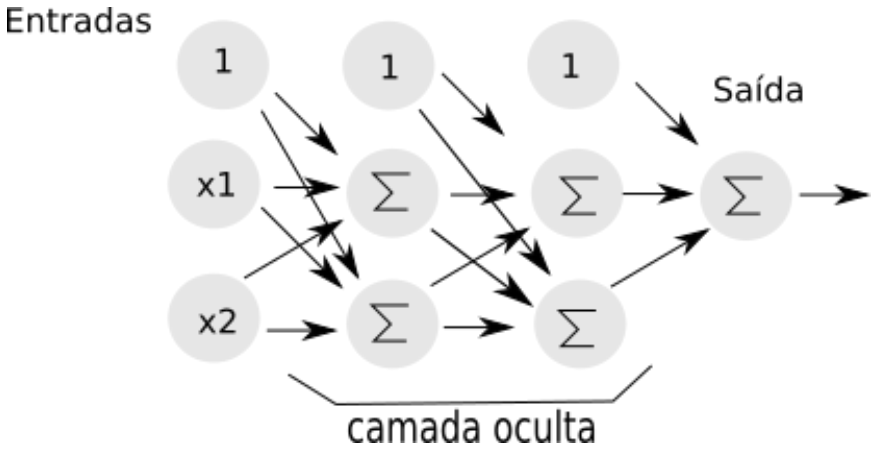

Figura 2.3: Múltiplas Camadas

2.1.3 Funções de Ativação

Conforme observado pela Equação 2.1, para obtermos o valor de saída de um nó, somamos as entradas multiplicadas pelos pesos, acrescentamos um viés e aplicamos a função de ativação. Inspirada na ideia de que um neurônio biológico pode ou não estar ativo e posteriormente sendo corroborada pela necessidade da utilização de funções não lineares para assim poder descrever problemas mais complexos [10] que funções lineares possuíam limitações para fazer, as funções de ativação se tornaram uma parte importante das redes neurais. Sua presença gerou a possibilidade de resolução de problemas mais complexos, como problemas de visão computacional e processamento de linguagem natural.

Estas funções necessitam ser de fácil resolução do gradiente, conforme veremos a seguir, o

que nos gerou algumas funções amplamente utilizadas, como a função sigmóide logística:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}, \tag{2.2}$$

e a tangente hiperbólica.

$$\tanh(z) = 2\sigma(2z) - 1, \tag{2.3}$$

Tais funções são bastante usadas pela comunidade de aprendizado de máquina, a função sigmóide mapeia a entrada para um espectro entre 0 e 1, já a função tangente hiperbólica para um espectro entre -1 e 1. Porém, tais funções possuem limitações, para valores de entrada distante da origem, o gradiente será muito pequeno, dificultando o processo de aprendizagem. Desta forma outras funções de ativação foram propostas, como a unidade de retificação linear (ReLU)

$$\text{relu}(z) = \max(0, z), \tag{2.4}$$

função que também possui caráter não linear, desta forma podendo descrever problemas mais complexos. Esta função é usada na maioria dos casos por possuir baixo custo computacional e possuir bons resultados empíricos, mesmo podendo apresentar problema com gradientes próximos ao zero [11]. Uma outra função bastante utilizada é a função softmax,

$$\text{softmax}(z) = \frac{\exp(z)}{\sum_i \exp(z_i)}, \tag{2.5}$$

sendo um tipo de função sigmóide, costuma ser mais utilizada na camada de saída da rede neural em casos de rede de classificação. A função softmax transforma as saídas para um escopo entre 0 e 1, nos fornecendo a probabilidade da entrada estar em determinada classe das classes de resposta [12].

2.1.4 Funções de Custo

A função de custo é a função utilizada para conseguir mensurar o quanto a rede neural está em desacordo em relação a saída desejada. Desta forma, utilizando-a conseguimos usar uma entrada na rede neural e quantificar a proximidade, ou não, em relação a saída desejada. Com ela chegamos ao valor do erro que possuímos na rede neural, assim conseguindo traçar estratégias para minimizá-lo.

As funções custo utilizadas no trabalho foram o erro quadrático médio (MSE):

$$L = \frac{1}{N} \sum_{i=1}^{N} (t_i - y_i)^2, \tag{2.6}$$

e entropia cruzada binária:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\Big(t_i log(y_i) + (1-t_i) log(1-y_i)\Big). \tag{2.7}$$

onde os valores y_i indicam a saída desejada pela rede e os valores t_i são as predições que a rede forneceu como saída.

Estas funções definem a forma da função custo e são importantes para conseguir uma melhor otimização do resultado da rede neural. A função de entropia cruzada binária penaliza bastante o erro de uma rede neural de classificação, possuindo um bom potencial de aplicação para a nossa aplicação.

2.1.5 Otimizadores

Os algoritmos otimizadores são os responsáveis por manipular o peso de influência de cada nó no resultado final, de forma a diminuir o erro entre as previsões feitas pela rede neural e a saída real. Considerando que a função de custo nos quantifica o erro do desempenho da rede neural em relação ao valor real, podemos calcular o gradiente desta função, assim descobriremos para qual direção e sentido a função possui um maior incremento, e invertemos esse sentido. Assim conseguimos obter a direção e sentido de como minimizar a função custo, o que fará obtermos um melhor desempenho da rede neural.

O gradiente pode ser obtido de duas formas, utilizando o método numérico ou o método analítico, porém o método numérico acaba sendo muito custoso, desta forma o cálculo analítico se prova mais eficiente. O método analítico se baseia no algoritmo de retro-propagação [13], permitindo o cálculo recursivo do gradiente com base na regra da cadeia. Ele se baseia no conhecimento prévio da derivada das funções de ativação de modo a atualizar os pesos da rede neural a partir do erro calculado pela função custo na saída da rede

$$\delta^L = \frac{\partial L}{\partial z^L} = \frac{\partial L}{\partial y^L} f'(z^L), \tag{2.8}$$

se propagando pelas camadas internas

$$\delta^l = \frac{\partial L}{\partial z^l} = \frac{\partial L}{\partial y^l} f'(z^l), \tag{2.9}$$

em que o termo

$$\frac{\partial L}{\partial y^l} = \sum_{i,j} w^l \frac{\partial L}{\partial z^{l+1}} \tag{2.10}$$

demonstra a influência dos pesos na variação do custo. Calculado o custo, é possível atualizar os pesos de todos os nós da rede, de forma a melhorar o desempenho da mesma.

Porém, há mais uma variável para influenciar essa atualização dos pesos, o termo η

$$w^l[t+1] = w^l[t] - \eta \frac{\partial L}{\partial w^l[t]}. \tag{2.11}$$

conhecido como taxa de aprendizado, é um valor importante para o aprendizado da rede, visto que valores muito grandes podem refletir em uma piora de rendimento da rede, ao invés de uma melhora, e valores muito pequenos podem refletir uma grande demora para obtenção de resultados significativos[14].

O treinamento da rede é dividido em épocas, sendo cada época uma passagem ao longo de todos os dados do conjunto de treinamento. Também é comum tais épocas em *batches*, que seriam pedaços destas épocas, onde a atualização dos pesos da rede são feitas baseadas no desempenho dela nestes pedaços. Isto ocorre pois caso a atualização seja feita para cada elemento do conjunto de dados, pode acarretar em atualizações enviesadas por um dado muito fora da curva acarretando numa piora dos resultados da rede, além de aumentar o tempo necessário para treiná-la, visto que a atualização também consome tempo, e ao atualizarmos a cada elemento do conjunto de dados, utilizaremos mais tempo para treinar a rede.

2.1.6 Curva de Aprendizado

O estudo das curvas de aprendizado é importante para avaliar e compreender o desempenho de modelos de aprendizado de máquina, permitindo entender a capacidade da rede de generalizar o problema. Para isto, realiza-se uma análise da acurácia da rede ao longo do seu treinamento utilizando um conjunto de dados para treinamento e um para teste. Esta análise nos permite diagnosticar a possibilidade da rede estar em um estágio de *overfitting* ou *underfitting*[14].

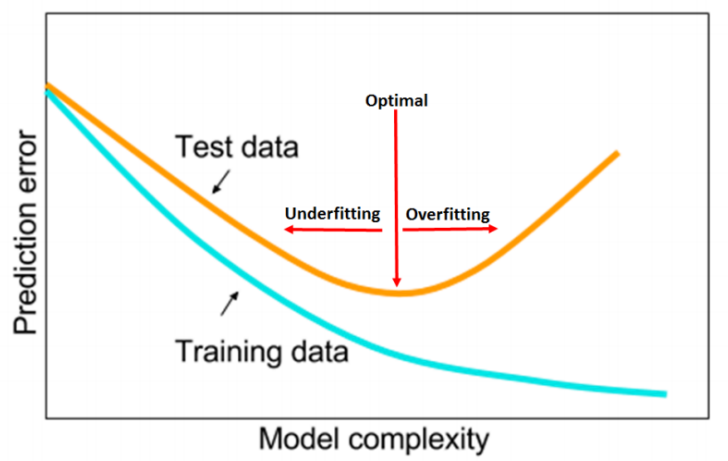

Figura 2.4: Curvas de aprendizado. Adaptado de [14].

O *overfitting*, ao pé da letra sendo traduzido para ajuste em excesso, desbalanceia o funcionamento da rede e pode ser analisado como o treinamento tão intenso dela em um dado conjunto de dados, que a rede começa a diminuir a capacidade de generalização. Neste momento, podemos

perceber que a sua acurácia, apesar de continuar melhorando quando analisamos em relação ao conjunto de dados de treinamento, começa a apresentar uma diminuição dela quanto testamos o desempenho da rede nos dados de teste [14], conforme identificado na Figura 2.4.

Já o *underfitting* é quando o modelo não é capaz de reduzir o erro de predição nem para o conjunto de dados de treinamento, nem para o conjunto de dados de teste. Desta forma, ou ele não é poderoso o suficiente, ou não foi treinado o suficiente para se detectar os padrões presentes no conjunto de dados [14], conforme podemos notar na Figura 2.4.

O momento em que as taxas de erro e validação se aproximam, estando próximas ao valor mínimo de validação, é o ponto ótimo de treinamento. Desta forma podemos identificar que o modelo esta generalizando bem para os parâmetros de treinamento escolhidos e a quantidade de épocas para treinar o modelo antes de acarretar no *overfitting*.

3 CODIFICAÇÃO DE VÍDEO

Este capítulo traz os conceitos básicos de codificação de vídeo para um melhor entendimento deste trabalho. Embora um estudo completo de codificadores de vídeos esteja fora do escopo deste trabalho, eles podem ser encontrados em [15] [16] [17]. Este capítulo trata dos principais temas de compressão de vídeo.

3.1 VÍDEO

Um vídeo são cenas capturadas espacialmente e temporalmente de forma contínua, podemos representá-lo como um *array* de três dimensões de *pixels*, sendo duas espaciais e uma temporal. As duas dimensões espaciais representam a imagem e a temporal a transição das imagens conforme podemos notar na Figura 3.1 [15].

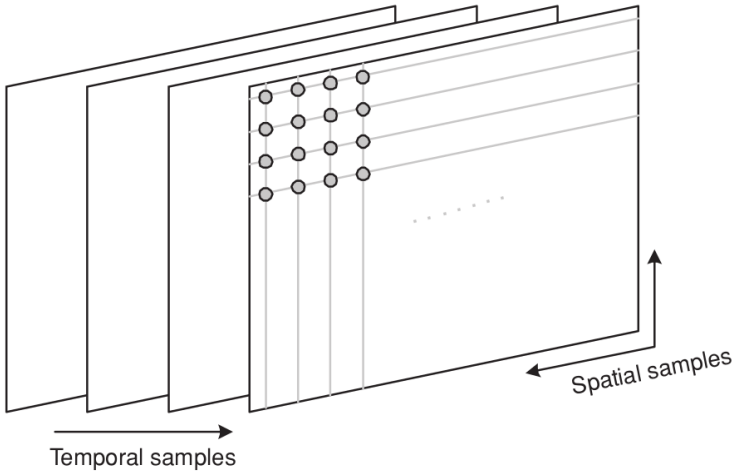

Figura 3.1: Representação de Vídeo adaptado de [15]

Os pixels podem ser representados de formas diferentes. Caso seja uma imagem monocromática, eles serão representados por um valor, a luminância, que indicará a escala de cinza. Já para imagens coloridas, ele precisará representar as cores, e para isso precisará de mais informação.

Uma forma de representar as cores é usando a escala RGB, onde cada pixel é descrito por três valores, o vermelho, o azul e o verde. A escala de cada uma dessas cores varia de 0 a 255, desta forma o vermelho seria representado por (255,0,0), o verde por (0,255,0) e o azul por (0,0,255). As outras cores são formadas por combinações destas cores, utilizando intensidades diferentes, como por exemplo, o amarelo é o (255,255,0). Assim, notamos que na representação RGB utilizamos 3 bytes para representar cada pixel [15].

Outra forma comum de representar é utilizando o sistema YUV, onde Y é conhecido como

luminância e UV como crominância, neste sistema Y é calculado como a soma ponderada dos três valores de RGB, enquanto as componentes de crominância são calculadas como a diferença entre cada componente RGB e a luminância. Desta forma podemos calcular conforme as equações 3.1

$$
\begin{aligned}
Y &= 0.299R + 0.587G + 0.144B \\
U &= 0.564(B - Y) \\
V &= 0.713(R - Y)
\end{aligned}
\tag{3.1}
$$

$$
\begin{aligned}
R &= Y + 1.402V \\
G &= Y - 0.344U - 0.714V \\
B &= Y + 1.772U
\end{aligned}
$$

O espaço YUV é comumente utilizado para a representação de imagens pois com ela podemos sub amostrar os componentes de crominância, o que ocorre nas representações YUV 4:2:2 e 4:2:0, onde não há a necessidade de um byte para cada canal, diferente da YUV original, também chamada de YUV 4:4:4. Esta sub amostragem é possível pois o sistema humano de visão (HVS) é menos sensível a cor do que a luminância, desta forma a crominância pode ser representada utilizando menos dados. Para o observador casual, não há diferença notável de qualidade entre uma representação RGB e uma representação YUV com crominância sub amostrada [15].

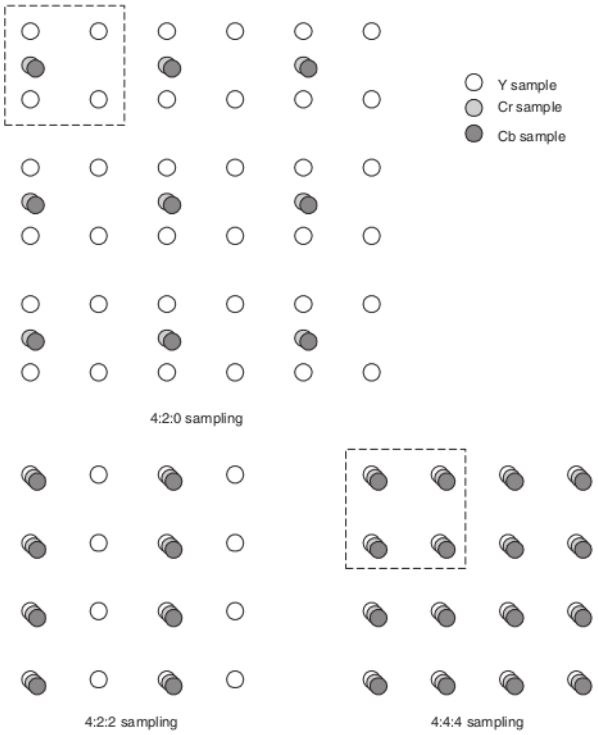

Figura 3.2: Formas de representação de vídeo no formato YUV, adaptado de [15]

Na representação 4:2:2, a crominância acaba sendo sendo sub amostrada, utilizando metade da taxa de amostragem da luminância no eixo horizontal, reduzindo o tamanho do video em um

terço. Já na representação 4:2:0, a sub amostragem ocorre tanto no eixo horizontal quanto no eixo vertical, o que acarreta na redução de metade do tamanho do vídeo, conforme observado na Figura 3.2. O modo 4:2:0 também referenciado como 12 bits por pixel, assim demonstra sua redução, em vez de utilizar os 24 bit por pixels usuais do YUV tradicional ou até o RGB.

3.2 CODIFICAÇÃO DE VÍDEO

Codificadores de vídeo buscam explorar redundâncias no vídeo, podendo ser entre frames, no próprio frame e até nos pixels [16]. Com essas técnicas o codificador tentará gerar um vídeo comprimido, tentando balancear entre qualidade do vídeo e tamanho final obtido para essa compressão.

Muitos tipos de dados contêm redundância estatística podendo gerar compressão sem perdas, desta forma os dados reconstruídos são uma cópia exata do original, porém, infelizmente, esse forma de compressão é bem limitada, possibilitando uma taxa de compressão de, aproximadamente, 3 a 4 vezes. Para atingir níveis de compressão maiores, precisamos de compressão com perdas, desta forma a reconstrução dos dados após a compressão não forma uma cópia exata dos dados originais, mas conseguimos atingir taxas de compressão bem maiores. Para tal utilizamos princípios de remoção de redundância subjetiva, removendo elementos da imagem, ou video, que não afetam significativamente a percepção do HSV [15].

3.2.1 Medida de qualidade objetiva

A percepção visual de uma cena é formada por complexas interações entre os componentes e o sistema visual humano, sendo ele os olhos e cérebro. Desta forma a percepção de qualidade de imagem se torna algo subjetivo, sendo influenciada pelo quão nítido as partes dela estão, ou até distorções na imagem. Porém, análises subjetivas dificultam a mensuração da qualidade de um vídeo, então desenvolvedores de compressão e processamento de vídeos utilizam uma métrica quantitativa para analisar a qualidade do vídeo. Dentre as várias métricas quantitativas propostas, a mais utilizada é a relação sinal-ruido de pico (PSNR).

O PSNR é uma medida logarítmica que depende do erro quadrático médio (MSE) entre a imagem original e a mesma pós compressão usando o quadrado do maior valor de sinal na imagem, onde n é o numero de bits por amostra da imagem:

$$PSNR_{dB} = 10 \log_{10} \frac{(2^n - 1)^2}{MSE}. \tag{3.2}$$

Com a Equação 3.2 notamos que ela pode ser calculada facilmente e de forma rápida, o que contribuiu para se tornar uma forma bem popular de comparar qualidade entre vídeos comprimidos e não comprimidos. Mesmo sendo a métrica mais usada para qualidade, possui seus oposito-

res. Os argumentos mais comuns são a necessidade da imagem original e a possibilidade de não retratar a percepção subjetiva da qualidade da imagem.

Um valor de PSNR alto usualmente indica uma qualidade maior, enquanto uma PSNR baixa uma qualidade pior. Porem, o valor PSNR não indica necessariamente uma qualidade subjetiva. Essa diferença entre os resultados objetivos e subjetivos se dá pois a PSNR possui pesos iguais para todos os pixels da imagem, sendo a variação em um único pixel, o suficiente para variar bastante no seu valor, o que pode não se refletir numa análise subjetiva da qualidade. Por exemplo, borrar uma imagem numa área que o observador não vai focar, pode degradar muito a PSNR, porém o observador humano pode nem notar tal diferença, ao mesmo tempo, distorções como o borramento da área mais focada da imagem pelo observador, pode acarretar numa significava queda de qualidade subjetiva e ainda obter uma PSNR melhor que o caso anterior [15].

3.2.2 Quantização

Codificadores podem usar um medida de quantização escalar, que tem como objetivo mapear um espaço de valores X para um espaço menor de valores Y. Assim deve ser possível representar os pixels usando menos bits, por diminuir a o conjunto de valores que os pixels podem assumir. A quantização escalar mapeia um valor de entrada para um valor de saída quantizado.

Desta forma, o novo valor do pixel será calculado da seguinte forma:

$$\begin{aligned} FQ &= round(\frac{X}{QP}) \\ Y &= FQ.QP. \end{aligned} \tag{3.3}$$

Desta forma, para maiores valores de QP, maior quantidade de números terão sua divisão dada com números decimais, e maior quantidade de números serão arredondados para os mesmo valores. Logo diminuímos o conjunto de números e geramos uma redundância maior de dados, podendo gerar uma maior compressão. Notamos também, que por existir um arredondamento, essa aplicação acaba sendo um processo com perdas, pois não é reversível. [15]

3.2.3 Unidades de Codificação

Codificadores de vídeo, em geral, dividem cada frame do vídeo em blocos. No H.264 [15], esse bloco unitário tem tamanho fixo de 16x16 pixels, sendo denominado como macrobloco. Já no H.265, eles possuem tamanho variável com tamanho máximo de 64 x 64, que é denominado como coding tree unit (CTU) [17].

Cada CTU pode ser sub-dividida em outras unidades de codificação (CU), dividindo cada lado pela metade, desta forma a CTU sai de 64 x 64 pixels de tamanho e se torna 4 CU de tamanho 32 x 32. Isso pode ocorrer recursivamente até o tamanho de 8 x 8 pixels.

Nas folhas desta árvore, ou seja, na menor divisão escolhida, são aplicadas a predição, trans-

formadas, quantização e codificação de entropia [15]. A decisão de sub-dividir, ou não, uma CU não é padronizada, diferentes codificadores têm algoritmos diferentes de tomada de decisão. O codificador usado neste trabalho, o HM 16.20, que tem como referência o H.265, testa as possibilidades com a divisão e escolhe o melhor resultado obtido comparando com e sem a ocorrência da divisão. Por testar todas as possibilidades, ele possui a vantagem de conseguir codificar o vídeo da melhor forma dentro das suas possibilidades, conseguindo atingir uma qualidade boa de compressão. Ao mesmo tempo, acaba possuindo uma complexidade grande, fazendo ele ser um algoritmo lento e que necessita de bastante capacidade computacional.

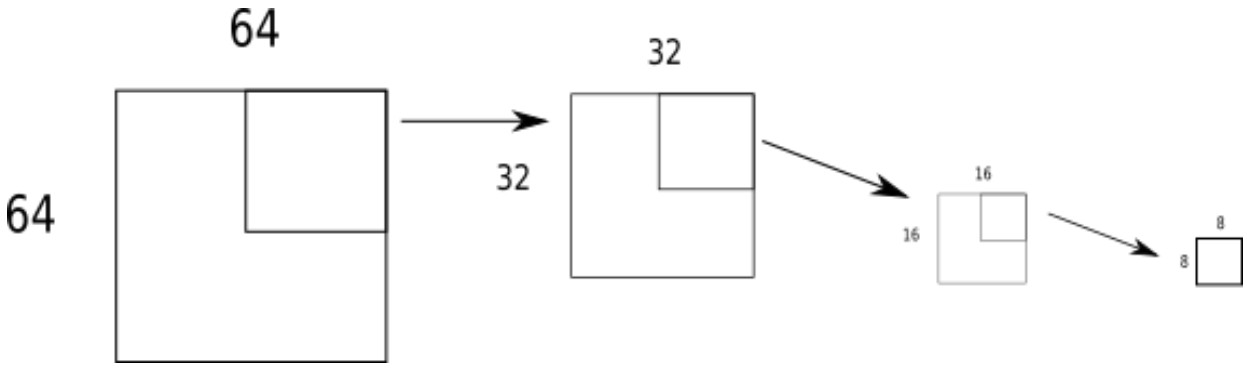

Figura 3.3: Split da CU

4 DESENVOLVIMENTO

O método proposto neste trabalho é a utilização de redes neurais para atuarem na decisão do codificador de fazer a divisão , ou não, das CTUs em CUs menores. Desta forma, foram criadas 3 redes neurais, cada uma responsável por decidir em uma profundidade, se o split será feito. Para tal, utilizamos uma versão modificada do HM 16.20, que se baseia no padrão H.265, para conseguirmos tanto gerar nossos dados, quanto rodar nossas simulações para obtermos dados e conseguirmos comparar o desempenho do nosso método em relação ao modo sem aceleração. Para este trabalho, escolhemos utilizar apenas o canal de luminância do vídeo como informação para decidirmos o split, não utilizando as crominâncias. O *framework* utilizado neste trabalho foi o *Tensorflow* [1], utilizando um computador com 24 GB de memória RAM, 6 GB de memória SWAP e placa de vídeo (GPU), para efetuar o treinamento das redes, GTX 970 G1 Gaming com 4 GB de memória RAM. O código de referência do trabalho se encontra no link https://github.com/trosemberg/Deeplearning_Video_Compressing

4.1 CONJUNTOS DE DADOS

Utilizamos o próprio HM 16.20 para fornecer os dados que foram utilizados para treinar as redes neurais. Comprimimos seis vídeos de tamanho 1920 por 1080 e formato YUV 4:2:0, com quatro valores diferentes de QP para que conseguíssemos os dados de treinamento. Com a compressão dos vídeos Crowdrun, DucksTakeOff, OldTownCross, ParkJoy, RushHour e Sunflower com QPs de valores 22, 27, 32 e 37 conseguimos gerar os dados necessários para treinar nossas redes.

As configurações para gerar os dados e treinar as redes se baseiam nas condições de teste comum (CTC) do padrão HEVC 265, sendo um acordo para padronizar os testes de compressão. Nele ficam definidos os valores de QP utilizados, vídeos para validação dos resultados entre outras configurações, as quais foram seguidas neste trabalho. Desta forma a replicação dos resultados obtidos nesse trabalho se tornam possíveis por haver um padrão de testes.

Para cada vídeo comprimido, obtivemos arquivos com as informações de split ou não. Estes arquivos possuem os seguintes dados: o nome do frame em análise, a profundidade de análise, sendo Depth 0 sendo referente a decisão de split para CTU de tamanho 64 por 64, Depth 1 para CUs de tamanho 32 por 32 e Depth 2 para CUs de tamanho 16 por 16, conforme podemos visualizar na Figura 4.1.

Cada vídeo citado acima possuía 500 frames, desta forma cada vídeo gerou 1500 destes arquivos sendo 500 arquivos para cada profundidade. Dentro destes arquivos se encontra a informação de se deve fazer a quebra das CUs em menores, conforme podemos ver na Figura 4.2, onde conseguimos a posição inicial de cada CU em relação a altura e largura do frame analisado.

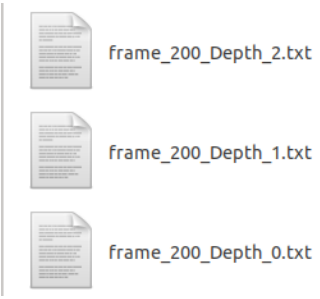

Figura 4.1: Formato dos arquivos gerados pelo H.265 ao gerar o dados para treinar as redes neurais.

```
CU (   0,1792) - QP = 29 - SPLIT = 0
CU (   0,1856) - QP = 29 - SPLIT = 0
CU (  64,   0) - QP = 29 - SPLIT = 0
CU (  64,  64) - QP = 29 - SPLIT = 0
```

Figura 4.2: Formato dos dados gerados pelo H.265 para identificar o split.

4.2 CARREGAMENTO DE DADOS

Durante a execução do trabalho, o tratamento de quantidades significativas de dados mostrou-se um dos maiores entraves para a experimentação efetiva de diferentes modelos de redes neurais. Para o treinamento da rede neural responsável pela decisão de split com profundidade 0, o conjunto de dados tinha tamanho aproximado de 6 milhões de entradas, para a profundidade 1 aproximadamente 25 milhões de entradas e para a profundidade 2, 100 milhões de entradas. Cada entrada é uma CU, desta forma, cada entrada é uma matriz do tamanho da CU para aquela profundidade, ou 64 por 64, ou 32 por 32, ou 16 por 16, desta forma transformar todas essas CU dos vídeos em imagens para depois utilizá-las como entrada da rede neural não pareceu ser a estratégia mais eficiente.

A estratégia inicial para contornar este problema foi tentar gerar uma tabela onde cada linha fosse o vetor que seria utilizado como entrada da rede neural, para tal tentamos gerar uma tabela de metadados onde possuíssemos as informações sobre de qual vídeo a CU pertencia, o frame deste vídeo, sua posição inicial em relação a altura e largura e QP de compressão desejada, conforme conseguimos identificar na Figura 4.3. Com isso bastaria aplicar uma função que mapeasse todos essas dados e nos gerasse a matriz de dados referente a CTU. Essa abordagem se mostrou bastante ineficaz ao analisarmos que desta forma teríamos aproximadamente 6 milhões de entradas, onde cada uma delas teria $64x64$ bytes o que nos forneceria aproximadamente 24.576.000.000 bytes de informação, ou seja, 24,5 GB para manter essa tabela em memória em seu estado final, desconsiderando o acréscimo de informação com o cabeçalho da estrutura para o computador entender esses dados. Esta informação que seria ainda maior durante o processo de gerar essa tabela pois precisaria da tabela de metadados em memória e também os vídeos para fazer esse mapeando, onde os 6 vídeos juntos dão um total de 9,3 GB.

Desta forma, foi necessária a utilização de outra abordagem para conseguir gerar os dados a

serem utilizados no treinamento da rede neural. A estratégia então adotada foi de utilizar uma tabela metadados para cada profundidade, onde com elas mapeamos as informações que serão usadas de entrada para as redes neurais, identificando o vídeo onde está esta informação, o frame e sua posição inicial em relação a altura e largura, tabela demonstrada na Figura 4.3. A partir disso nossa abordagem foi gerar uma classe que fosse capaz de processar essas dados no formato de metadados e transformá-los no formato desejado para o treinamento das redes neurais, porém somente em momento de treinamento, ou seja, as redes passariam a receber como entrada para seu treinamento a tabela de metadados e a classe seria responsável por, quando a rede neural necessitasse utilizar novos dados para treinamento, somente os dados do *batch* que fosse ser treinado seriam convertidos para o vetor de informação de entrada. Desta forma, mantendo em memória durante o treinamento apenas a tabela de metadados e os vídeos, pois a conversão dos metadados para os dados das CU foram feitas em tempo de execução do treinamento das redes neurais.

```
Informacoes
```

	Origem	Frame	Height	Width	QP	Split
0	SunFlower	338	0	0	22	1
1	SunFlower	338	0	64	22	0
2	SunFlower	338	0	128	22	1
3	SunFlower	338	0	192	22	0
4	SunFlower	338	0	256	22	1

Figura 4.3: Tabela com os Metadados que foram usados de entradas nas redes neurais

Para isso foi criada uma classe DataGenerator, ilustrado na Figura 4.4, que foi responsável por fazer a tratativa dos metadados, transformando-os nas entradas corretas para a rede neural. Para utilizar essa classe, primeiro precisou-se abrir todos os arquivos de vídeo no programa, desta forma mantendo-os em memória. Depois foi feita a criação de um dicionário em que a chave do dicionário é como o nome de cada vídeo estava escrito na tabela de metadados, assim quando olhássemos uma linha da tabela, conseguiríamos identificar a qual vídeo aquela linha estava se referindo.

O próximo passo foi, utilizando a informação do frame a que CU pertence, juntamente à informação de qual a sua posição inicial de altura, largura e do tamanho da CU, que é padrão para cada profundidade, varrer todo o vídeo até achar estes dados. Com isso, o DataGenerator se tornou capaz de interpretar cada linha da tabela metadados para encontrar, dentro do vídeo certo, a matriz de dados que é a CU desejada e isso somente ocorria no momento de carregar os dados para treinamento, o que se tornou uma solução relativamente leve em termo de consumo de memória, pois somente geraria os dados das CUs que seriam usadas no treinamento, no momento de utilizá-

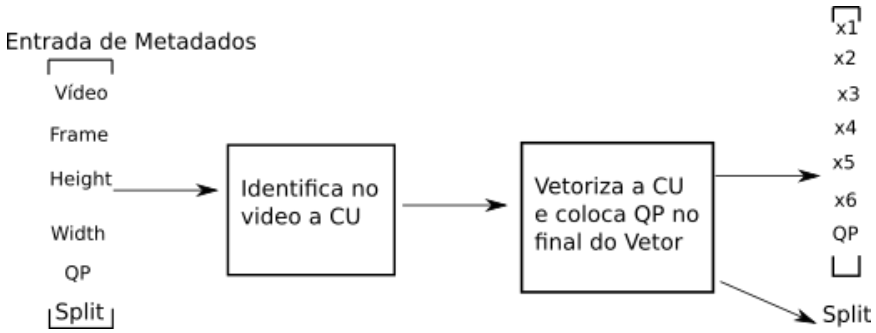

Figura 4.4: Funcionamento do DataGenerator

las. Por opção de estrutura da rede neural, foi decidido usar como saída do DataGenerator um vetor com os dados referentes a CU, acrescido de uma posição com a informação do QP o qual se deseja que aquela CU seja comprimida, em vez de utilizar a CU no formato matricial acrescida da informação do QP.

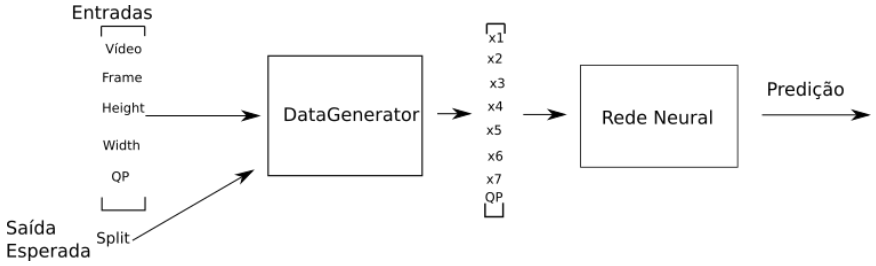

Figura 4.5: Fluxo Habitual de dados para treinamento da Rede Neural com DataGenerator

Foi testada, enquanto se decidia o design da classe DataGenerator, a possibilidade de manter os vídeos em disco e somente abri-los no momento necessário, porém com a necessidade de fazer muitos acessos ao disco (mínimo de 6 milhões para CU de 64 por 64), este design se tornou extremamente lento e custoso para ser executado.

Apesar de essa solução ter proporcionando uma maior facilidade para o carregamento de dados do que gerar um arquivo em disco para CU, essa solução também não foi a solução ótima para a resolução do problema pois apresentou um gargalo importante, o tamanho da memória do computador. Durante o treinamento da rede neural para classificação baseando-se em CUs de tamanho 16 por 16, a tabela de metadados possuía aproximadamente 100 milhões de entradas, juntamente a manter os vídeos em memória (cada vídeo possuía aproximadamente 1,6 GB de tamanho) acarretou na utilização de aproximadamente 26 GB de memória RAM do computador, o que é bastante e se torna um limitante pois nem todo o computador possui essa quantidade de memória.

Desta forma, o modo de abordar o treinamento das redes neurais do ponto de vista do carregamento de dados foi diferente do habitual. Conforme podemos identificar na Figura 4.6, o carregamento de dados no formato habitual consiste em fazer as manipulações de dados necessária antes de iniciar o treinamento da rede neural, desta forma passando como entrada para a rede neural o dado já no formato correto para efetuar o treinamento. Na nossa abordagem precisamos

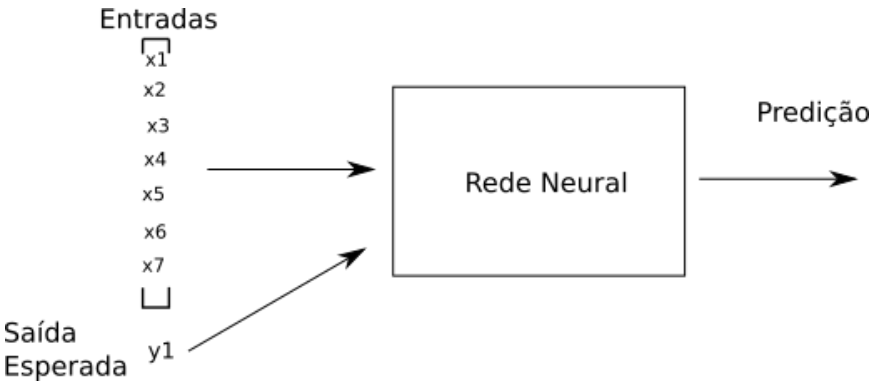

Figura 4.6: Fluxo Habitual de dados para treinamento da Rede Neural

passar para a rede neural metadados e criar uma forma de ela lidar com esses metadados para fazer o seu treinamento, conforme podemos notar na Figura 4.5.

4.3 TREINAMENTO

Para obtenção das redes capazes de detectar padrões e determinar o split, treinaram-se algumas arquiteturas diferentes utilizando como entrada os metadados referidos na seção anterior. Dentre as arquiteturas testadas, variando as funções de ativações, estrutura da rede neural e otimizadores, os melhores resultados obtidos foram usando modelos onde a saída de uma camada é usada para alimentar a entrada da próxima camada. Para a rede neural de profundidade zero, geramos um modelo com três camadas ocultas usando RELU e 30, 20 e 10 nós respectivamentes, e uma camada de sáida usando Sigmóide Logística. Para a rede neural de profundidade um, geramos um modelo com três camadas ocultas usando RELU e 50, 30 e 10 nós respectivamentes, e uma camada de sáida usando Sigmóide Logística. Por último, para a rede neural de profundidade dois, geramos um modelo com três camadas ocultas usando RELU e 30, 30 e 10 nós respectivamentes, e uma camada de sáida usando Sigmóide Logística.

Todas as redes utilizaram como função de custo, a função de entropia cruzada binária. Com tais configurações alcançamos acurácia nos dados de validação de 63% , 82% e 91% para as redes das profundidades 0, 1 e 2, respectivamente. Como padrão, cada rede foi treinada por 5 épocas, e a duração do treinamento das redes foi um limitante para testar novas possibilidades, visto que para o treinamento das redes de profundidade 0, 1 e 2, cada época demandou aproximadamente 8h, 10h e 13h respectivamente, assim nos demandando 40h para o treinamento da primeira rede, 50h para o treinamento da segunda rede e 62h para o treinamento da terceira rede. Desta forma não foi possível testar muitas possibilidades de configuração das redes por demandar muito tempo. Para as três redes apresentadas com melhor acurácia, na última época o valor da função custo tanto no conjunto de dados de validação quanto no conjunto de dados de treinamento continuavam diminuindo, porém a taxas muito pequenas em relação as épocas anteriores, o que pode indicar uma aproximação do ponto ótimo de treinamento para estas redes.

4.4 PREDIÇÃO

Para obtenção dos resultados, utilizamos o H.265 utilizando arquivos de entrada para o programa com o formato e informações apresentados nas Figuras 4.7 4.8.

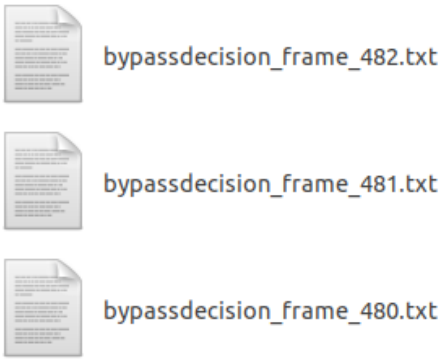

Figura 4.7: Arquivos de Entrada para aceleração da compressão

```
CU ( 192, 192) - 0
CU ( 192, 256) - 1 0 0 0 0
CU ( 192, 320) - 1 0 1 1 0 0 0 0 0 0 0 0 0
```

Figura 4.8: Estrutura do Arquivo de Aceleração da compressão

Desta forma, para cada frame de cada vídeo para cada valor de QP, foram utilizadas as redes neurais para predizer qual o valor do split para aquelas CU. Estas informações foram salvas em arquivos com o nome de *bypassdecision_frame_(número correspondente ao frame do vídeo).txt* e estes arquivos foram usados como entrada para o H.265, desta forma o H.265 utiliza tais informações para realizar, ou não, o split. Caso o resultado seja 1, o H.265 ignora os testes a serem feitos na CU e já efetua o split, desta forma economizando tempo durante a compressão. Caso o resultado de split seja 0, o H.265 não testará as possibilidades relacionadas a execução do split da CU, ele considerará que o melhor resultado a ser obtido está ao analisar aquela CU sem efetuar o split.

A estrutura do Arquivo de aceleração fornece a CU a ser analisada, sendo o primeiro valor a posição inicial dela em relação a altura e o segundo valor em relação a largura. A estrutura da sequência de 1s e 0s para cada CU significa a decisão de fazer ou não o split, desta forma, caso a decisão seja de fazer o split, são adicionados 4 novas posições no arquivo para indicar o resultado da análise da nova profundidade. A Figura 4.9 ilustra como está organizada essa estrutura, para ela usamos de exemplo o resultado da última linha da Figura 4.8. Com ela conseguimos identificar melhor o padrão apresentado dentro do arquivo de aceleração. Os números em vermelho representam a ordem em que o valor de split aparecerá no arquivo de bypass, seguindo a ordem da figura formamos que para a CU que se inicia na posição 192 verticalmente e 320 horizontalmente, as decisões de split são dadas, sequencialmente por 1 0 1 1 0 0 0 0 0 0 0 0 0, onde o primeiro valor representa a decisão de split para quando a CU possui tamanho 64 por 64 e os próximos 4

valores para a CU quando ela possui tamanho 32 por 32, e assim sucessivamente.

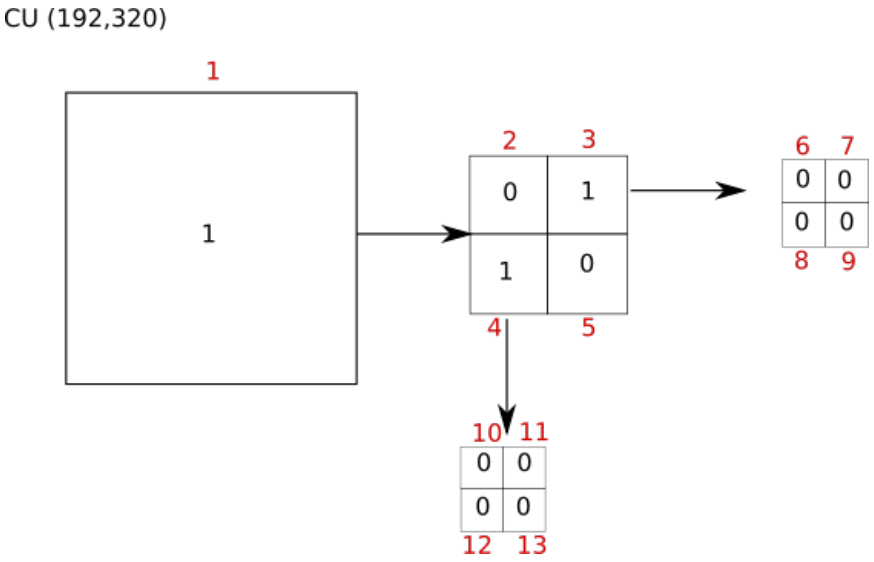

Figura 4.9: Estrutura do Arquivo de Aceleração da compressão

5 RESULTADOS

Baseando-se em toda a estrutura previamente descrita, rodamos o H.265 para comprimir 5 novos vídeos, com dimensões iguais aos utilizados para treino e também no formato YUV 4:2:0. Esses vídeos não estiveram presentes na etapa de treinamento, apenas na etapa final para adquirimos resultados. Os vídeos foram o *BasketballDrive*, *BQTerrance*, *Cactus*, *Kimono* e *ParkScene*, todos são definidos pela CTC, sendo vídeos de classe B, que são vídeos 1920x1080. Para fins de comparação de desempenho do trabalho, comprimimos estes vídeos no mesmo computador utilizando 3 formas diferentes, a primeira foi a compressão utilizando o modo padrão do HM 16.20 nos 4 valores de QP especificados. Depois foi feita a compressão deles a partir dos arquivos gerados pela análise das redes neurais produzidas neste trabalho.

O terceiro caso serve como parâmetro de desempenho mais próximo da rede neural para comparação. Neste caso, utilizou-se também a estrutura de aceleração predizendo o split, porém se baseando em um algoritmo aleatório. Desta forma, a decisão do split tinha chances iguais de dar ou não split. Esse algoritmo também acelera a compressão porém gerou um resultado pior do que o utilizando a rede neural para decisão, conforme podemos analisar nas Figuras 5.1, 5.2, 5.3, 5.4, 5.5 e na Tabela 5.1. Na tabela também foram acrescentados dados referentes a outro modo de utilização do HM 16.20, que também é responsável por acelerar a compressão do vídeo [18], utilizando um modo nativo da própria aplicação e também respeitando as regras da CTC.

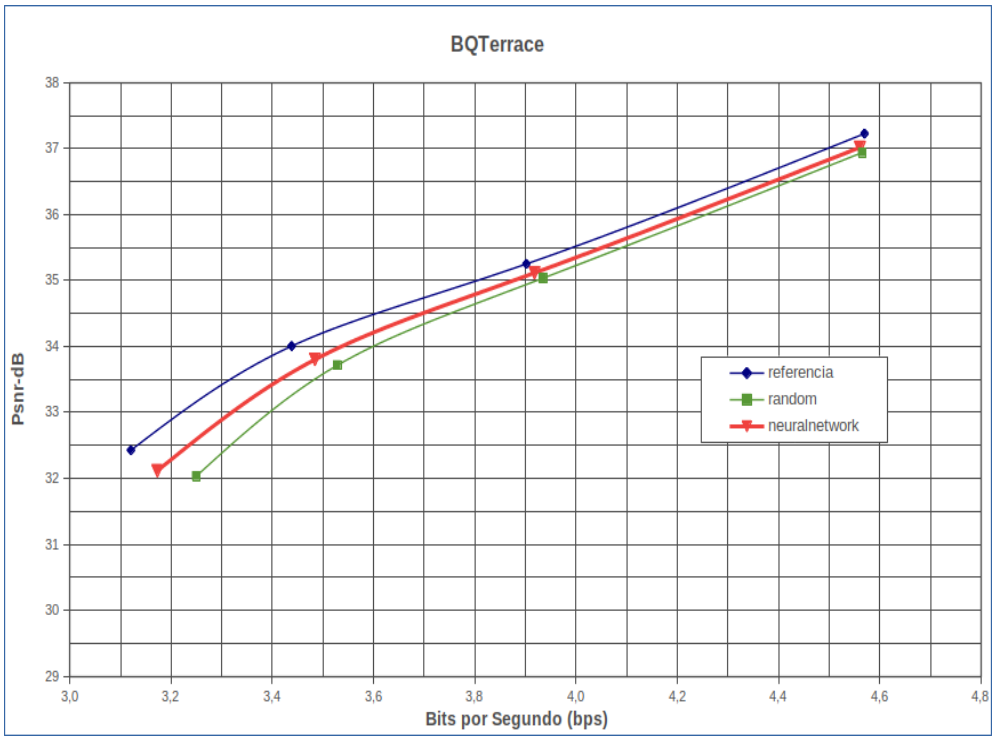

Figura 5.1: Comparativo entre o Padrão HM 16.20, Redes Neurais e decisão baseada na aleatoriedade para o vídeo BQTerrance.

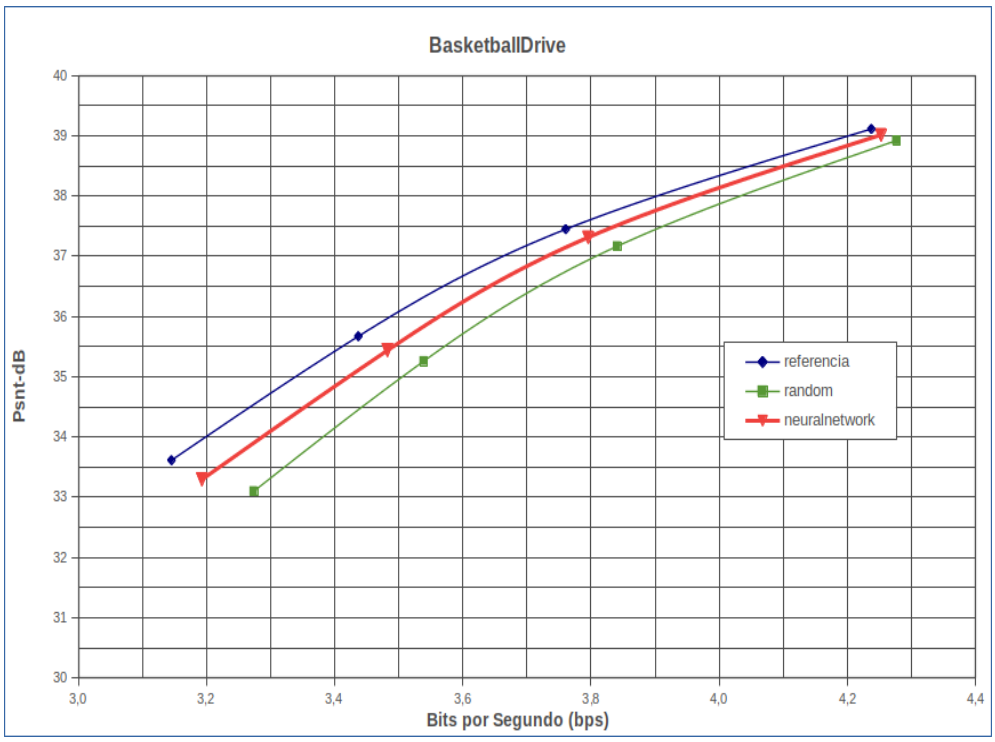

Figura 5.2: Comparativo entre o Padrão HM 16.20, Redes Neurais e decisão baseada na aleatoriedade para o vídeo BasketballDrive.

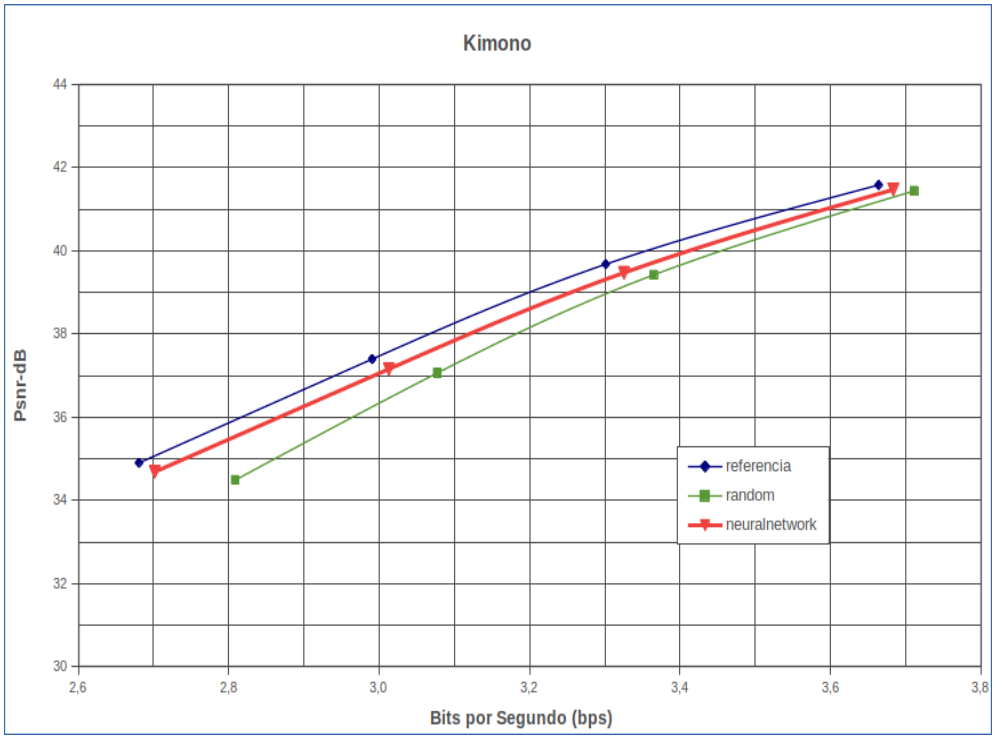

Figura 5.3: Comparativo entre o Padrão HM 16.20, Redes Neurais e decisão baseada na aleatoriedade para o vídeo Kimono.

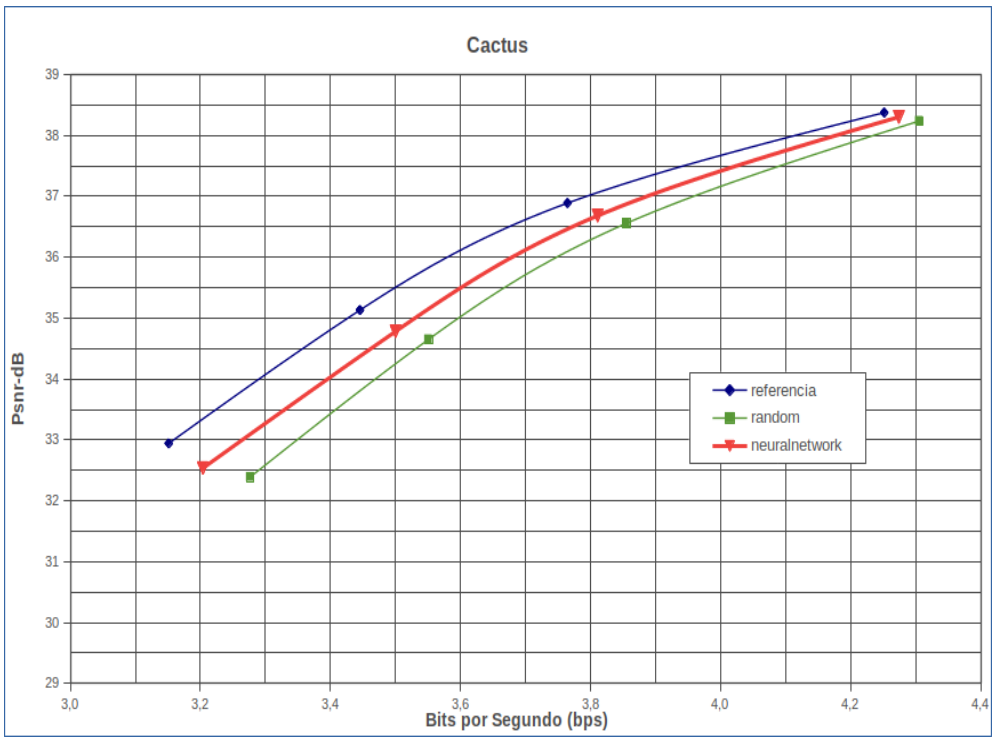

Figura 5.4: Comparativo entre o Padrão HM 16.20, Redes Neurais e decisão baseada na aleatoriedade para o vídeo Cactus.

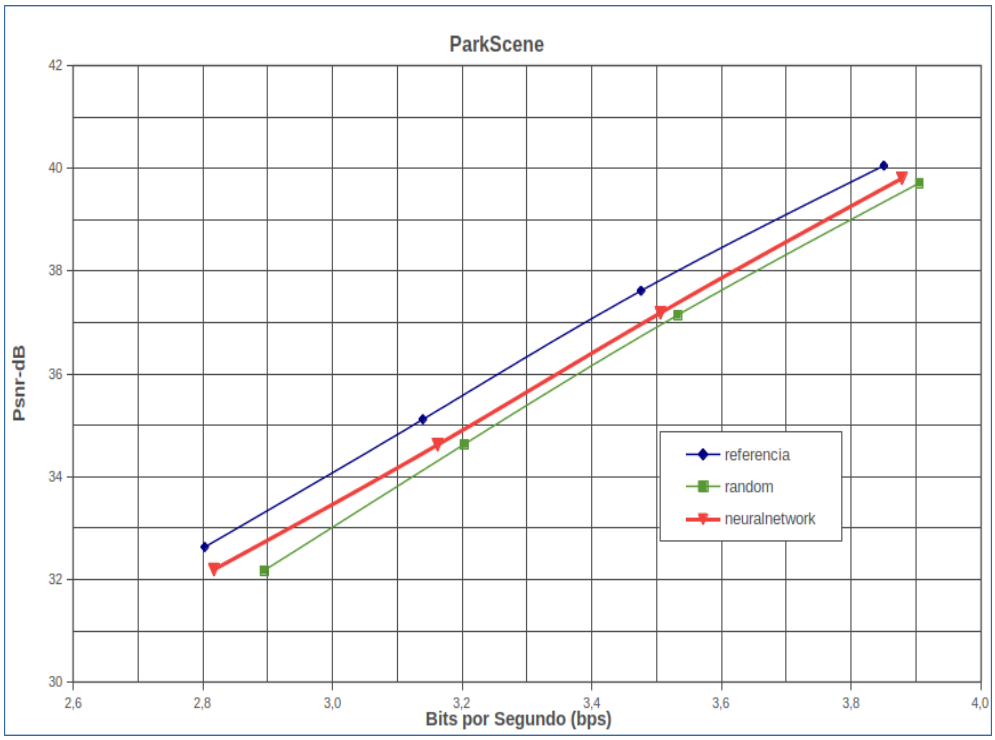

Figura 5.5: Comparativo entre o Padrão HM 16.20, Redes Neurais e decisão baseada na aleatoriedade para o vídeo ParkScene.

Para calcular o valor de *time saving* (TS) para cada video codificado, utilizamos a seguinte

Tabela 5.1: Comparação de desempenho usando BDR e TS dos métodos aleatórios, Rede Neural e *Fast-HM*

Video	Aleatório		Rede Neural		Fast-HM	
	BDR[%]	TS[%]	BDR[%]	TS[%]	BDR[%]	TS[%]
BQTerrace	34.7	77.7	19.2	53.1	2.5	55.1
BasketballDrive	41.5	77.5	18.2	56.9	1.5	45.6
Cactus	46.1	77.4	24.5	56.3	2.5	50.4
Kimono	31.3	77.5	12.7	68.1	1.5	51.1
ParkScene	33.6	77.5	21.8	60.8	1.9	59.4

fórmula [18]:

$$\Delta T_i = \frac{T_A - T_M}{T_A} \cdot 100\%, \tag{5.1}$$

onde temos que T_A representa o total de tempo para compressão do vídeo no formato base, que é a compressão do vídeo no formato padrão do HM 20.16 e T_M representa o total de tempo de codificação no formato testado. Para chegar ao valor apresentado na tabela, fizemos a média dos ΔT_i para os 4 valores de QP. Desta forma, um valor positivo de TS[%] indica a quantidade de tempo salvo usando tal técnica de compressão em relação a técnica usada como base.

Para medir a eficiência da técnica de compressão, utilizamos a distorção de Bjøntegaard (BD-rate) , no qual valores positivos indicam uma queda na eficiência da compressão em relação a técnica base [18].

Conforme podemos notar, os resultados obtidos pela rede neural apresentam um desempenho significativamente melhor do que os resultados a partir da decisão de split aleatória, o que indica que a rede neural foi capaz de detectar padrões de compressão para determinação do split, fazendo a decisão de split ser melhor do que a aleatoriedade. Também conseguimos valores melhores de TS do que a técnica de aleatoriedade. Atingindo assim a meta inicial do trabalho, que era de conseguir gerar redes neurais capazes de predizer o split das CUs melhor do que uma predição aleatória.

A nossa técnica proposta possui um desempenho significativamente pior que a técnica de *Fast-HM* em termos de BD-rate, porém atingindo resultados relevantes em termos de TS, onde para alguns vídeos apresentou resultados próximos, mas para outros como o Kimono, obteve um resultado significativamente melhor. Desta forma a nossa técnica se demonstra útil em aplicações que a prioridade seja comprimir o vídeo usando o padrão H.265 com economia de tempo, podendo ainda assim ter quedas na eficiência da compressão não tão altas se compararmos com a técnica de decidir o split aleatoriamente.

A técnica *Fast-HM* funciona de modo *in-loop*, ou seja, ele codifica de um certo modo e , depois, dado alguns critérios, decide se aquela codificação foi boa, caso não tenha sido, ele testa as outras formas. Tanto a técnica que se baseia na forma aleatória de decisão, quando a que utiliza a rede neural não fazem esse teste, desta forma sendo designadas fora do *loop*.

6 CONCLUSÃO

6.1 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho explora a implementação de redes neurais no contexto de acelerar a compressão de vídeo. Para tal, geramos redes neurais para cada decisão de split do codificador HM 16.20 que é baseado no padrão H.265, porém de forma de sistema aberto, ou seja, apenas tomamos uma decisão baseada na rede neural, que é a decisão de efetuar o split ou não e a aceitamos. Neste contexto foram geradas três redes neurais para este trabalho, cada uma atuando em uma área de decisão de split, e comparamos os resultados obtidos com resultados caso essa decisão fosse tomada de modo aleatório e caso não houvesse tal decisão.

Os resultados obtidos foram comparados com resultados usando a aleatoriedade para fazer essa decisão, ou seja, implementando uma forma de decidir se será feito o split com 50% de chance de ocorrer e também comparados com o modo *Fast-HM* do HM 16.20. Este modo alcança resultado consideravelmente melhores em termos de qualidade da compressão, porém nossa proposta consegue consegue gerar resultados relevantes utilizando muito menos tempo de compressão.

Uma estipulação de trabalho futuro a ser feito para alcançar melhores resultados é fechar o sistema, utilizando métricas para checar se o resultado obtido para aquela CU foi bom, e se não for testar outros resultados. Isso poderá gerar uma aumento no tempo de compressão, porém tal aumento pode vir em conjunto com uma melhor performance geral do sistema.

Outra área de melhora na implementação é a mudança da arquitetura da rede neural, para tentar obter melhores resultados e também a utilização de outras técnicas para tentar alcançar uma melhor performance da rede neural. Técnicas como utilizar a política de fit one-cycle [14], técnica que demonstra alcançar bons resultados no treinamento de redes neurais. Uma possibilidade de melhora mudando a arquitetura da rede, de forma que ao invés de transformar a entrada diretamente em vetor, seja possível tratá-la como imagem inicialmente. Desta forma, poderíamos utilizar de redes convolucionais, aproveitando redes já treinadas para classificação de imagens, usando a técnica de *transfer learning*, e adicionaríamos o QP para fazer parte da análise em uma camada oculta, ao invés de adicionar o QP na camada de entrada. Com isso não seria necessário buscar treinar uma rede neural do zero, além de buscar aproveitar uma rede neural que ja é capaz de identificar alguns padrões em imagens e tentar fazer ela detectar padrões nas nossas imagens.

REFERÊNCIAS BIBLIOGRÁFICAS

1 ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <http://tensorflow.org/>.

2 CISCO. *White paper: Cisco Visual Networking Index: Forecast and Trends, 2017–2022*. [S.l.], 2018. Disponível em: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>.

3 BOSSEN, F.; FLYNN, D.; SHARMAN, K.; SüHRING, K. *HM Software Manuals*. 2018. Software AHG working document. Disponível em: <https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/trunk/doc/software-manual.pdf>.

4 JCT-VC HEVC. *HM 16.20*. Disponível em: <https://hevc.hhi.fraunhofer.de/trac/hevc/browser/tags/HM-16.20>.

5 LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. SN -, v. 521, p. 436 EP –, May 2015. Disponível em: <https://doi.org/10.1038/nature14539>.

6 GéRON, A. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2017. ISBN 1491962291, 9781491962299.

7 MAGLOGIANNIS, I. *Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies*. IOS Press, 2007. (Frontiers in artificial intelligence and applications). ISBN 9781586037802. Disponível em: <https://books.google.com.br/books?id=vLiTXDHr_sYC>.

8 KOTSIANTIS, S. Supervised machine learning: A review of classification techniques. *Informatica (Ljubljana)*, v. 31, 10 2007.

9 MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, v. 52 1-2, p. 115–133, 1943.

10 LESHNO, M.; LIN, V. Y.; PINKUS, A.; SCHOCKEN, S. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, v. 6, n. 6, p. 861 – 867, 1993. ISSN 0893-6080. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0893608005801315>.

11 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 1097–1105. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

12 ALPAYDIN, E. *Introduction to Machine Learning*. 3. ed. Cambridge, MA: MIT Press, 2014. (Adaptive Computation and Machine Learning). ISBN 978-0-262-02818-9.

13 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In: RUMELHART, D. E.; MCCLELLAND, J. L.; GROUP, C. P. R. (Ed.). Cambridge, MA, USA: MIT Press, 1986. cap. Learning Internal Representations by Error Propagation, p. 318–362. ISBN 0-262-68053-X. Disponível em: <http://dl.acm.org/citation.cfm?id=104279.104293>.

14 SMITH, L. N. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018. Disponível em: <http://arxiv.org/abs/1803.09820>.

15 RICHARDSON, I. E. *The H.264 Advanced Video Compression Standard*. 2nd. ed. [S.l.]: Wiley Publishing, 2010. ISBN 0470516925, 9780470516928.

16 PEIXOTO, E. *Advanced Heterogeneous Video Transcoding*. Tese (Doutorado) — University of London, 2012.

17 WIEN, M. *High Efficiency Video Coding: Coding Tools and Specification*. [S.l.]: Springer Publishing Company, Incorporated, 2014. ISBN 3662442752, 9783662442753.

18 Zupancic, I.; Blasi, S. G.; Peixoto, E.; Izquierdo, E. Inter-prediction optimizations for video coding using adaptive coding unit visiting order. *IEEE Transactions on Multimedia*, v. 18, n. 9, p. 1677–1690, Sep. 2016. ISSN 1941-0077.